A Characterization of the Software Deployment Process and a Survey of Related Technologies

Antonio Carzaniga

September 18, 1997

Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci, 32
20133 Milano, Italy

Abstract

Software applications are no longer stand-alone systems. They are increasingly the result of the integration of heterogeneous collections of components, possibly distributed over a computer network. Different components can be provided by different producers and they can be part of different systems at the same time. Moreover, components change and evolve very rapidly, making it difficult to manage the whole system in a consistent way.

In this scenario, a crucial step of the software life cycle is deployment—that is, the activities related to the release, installation, activation, deactivation, update, and removal of components, as well as whole systems.

This paper presents a characterization of the deployment process together with a framework for evaluating technologies that are intended to address the software deployment problem. The framework highlights four primary factors that characterize the maturity of the technologies: process coverage; process changeability; interprocess coordination; and site, product, and deployment policy abstraction. A variety of existing technologies are surveyed and assessed against the proposed framework. Finally, we discuss promising research directions in software deployment.

1 Introduction

Software systems are no longer merely stand-alone applications. Increasingly, software systems are the integration of a collection of components, both executable and data, possibly dispersed over numerous heterogeneous networked nodes. Consider the relatively simple case of web browsers, which are in fact shells into which a variety of applications, such as document viewers and collaboration tools, can be fit to create the effect of an integrated system. Such "systems of systems" are created by integrating components from different organizations having different release schedules and different goals.

Software producers in this context no longer distribute complete systems. They must therefore find a way to deal with greater uncertainty in the environment within which their systems will operate. For example, before they can guarantee a successful installation, producers must be able to determine what components are available on a given node, as well as the configuration of those components. Additionally, since the components are produced by multiple organizations, they must be able to anticipate or react to updates to components that are not under their control. These issues are further magnified when the scale of the Internet—in terms of the vast numbers of producers and consumers, as well as the great distances involved—is taken into consideration.

Clearly, this is a difficult task that creates new challenges in the areas of release, installation, activation, de-activation, update, and removal of components. These activities constitute a large and complex process that we refer to as *software deployment*. The growing complexity of software systems mandates that software deployment activities be given more attention.

Recently, a number of new technologies have begun to emerge to address the deployment problem. Typical features offered by these technologies include system and configuration description, package and installation construction, automatic update delivery, and various network management capabilities. One emerging technology that has a particularly strong relationship to software deployment is content delivery. Systems such Castanet, PointCast, and recent enhancements to Microsoft and Netscape web browsers, make it possible to simplify and automate the transfer of raw data from one site to another. While software deployment includes a content delivery activity, in which the system to be deployed must be physically moved from the producer site to the consumer site, this is just one step that makes up the larger deployment process.

Despite the proliferation of deployment technologies, we have no clear understanding of the issues related to Internet-scale deployment, nor a way to evaluate the suitability of these technologies to address those issues. Thus, the purpose of this paper is threefold. First, we analyze and characterize the challenges and issues of software deployment. Second, we develop a framework for evaluating existing and proposed deployment technologies. This framework highlights four primary factors that characterize the maturity of the technologies: process coverage; process changeability; interprocess coordination; and site, product, and deployment policy abstraction. Finally, we provide a survey of a variety of systems and an assessment of them against the framework.

The main conclusion that can be drawn from this work is that no single existing system is able to comprehensively and coherently cover the full range of deployment activities in a way that factors out critical properties and characteristics of the entities involved in the deployment process. Moreover, it is not clear how different deployment technologies could be integrated to satisfy these requirements. Our framework and the results of our evaluation suggest areas where attention should be focused to guide future research and development activities in software deployment.

The paper is organized as follows. In the next section we present a motivating example to illuminate the issues in software deployment. From this example, we derive basic concepts and requirements. Section 3 presents our evaluation framework. A number of representative existing technologies are then evaluated against that framework in Section 4. We conclude with some thoughts about future research directions suggested by the evaluation.

2 Characterizing Software Deployment

Informally, the term *software deployment* refers to all the activities that make a software system available to its users. While this definition is reasonable and intuitively clear, the creation of an evaluation framework for software deployment technologies requires a more precise and comprehensive understanding of the nature and characteristics of software deployment. This section explores multiple facets of software deployment from several different viewpoints. In particular, we first introduce a motivating scenario that highlights the role of software deployment.[1] We then study and discuss software deployment from a structural viewpoint by identifying its basic constituent activities. Finally, we present requirements and constraints that must be taken into account in the development of the evaluation framework.

2.1 A Motivating Scenario

A medium-size organization uses an information system based over a wide-area computer network. The organization is hierarchically structured in a number of branches. Each branch has a local-area network with a branch server, several workstations, and personal computers. Every branch server is connected to a central server that resides at the organization's headquarters. The organization manages approximately thirty servers and a few thousand PCs and workstations. The headquarters as well as every single branch is connected to the Internet.

The organization's computing environment is host to a number of different applications. The central server and the branch servers manage the main database of the organization. PCs and workstations run database client applications, Web servers, mail systems, and common stand-alone tools for office automation (e.g., word processors and spreadsheets).

Each branch of the organization needs to consult a common set of files containing prices and product information. These files are administered by the organization's headquarters. However, to avoid traffic overhead, these files are replicated at every branch so that the requests can be handled locally. The files are updated once per day.

The organization acquires hardware and software components from a number of different providers. Some of the providers are connected to the Internet and offer electronic commerce facilities that provide on-line catalogs, negotiation of product features, and electronic purchase transactions. Some providers also offer on-line maintainance and automatic update services so that patches and updates can be automatically deployed by the provider, either in response to bug reports or to release new versions of products.

The configuration of the information system, both hardware and software, undergoes regular change. The organization's headquarters is responsible for maintaining the hardware and software configuration of systems and services that are shared among branches (e.g., the main database and the mail servers). Each branch is given freedom to customize its local environment to meet the specific needs of its business. Specifically, each branch is responsible for the configuration of branch-specific systems and information, such as database client applications, stand-alone tools, and user profiles. In every branch there may be applications installed on every machine as well as common branch-wide installations that can be shared by many users on a number of different machines, typically through an application server.

The obvious requirement of the organization is to minimize the overall cost of software management and deployment, where the cost function is determined by the loss or the down-time of functionality due to deployment activities, the additional labor spent in those activities, and the amount of computational and communication resources used for deployment. The organization policies also impose strict requirements over the deployment activities. For example, interference caused by deployment activities during business hours are not tolerated. Moreover, for security purposes, the organization must have full control over every deployment activity. This is achieved by means of strict authorization and test procedures.

2.2 Basic Terminology

The software deployment process may be defined as

> The delivery, assembly, and management at a site of the resources necessary to use a version of a software system.

We envision a network of computers partitioned into *sites*, where each site hosts a set of *resources*. In most cases, a site refers to a single computer. In general, however, a site may be a strongly coupled set of

[1] This example is partially inspired by the Digital Equipment Corporation's Project Gabriel [21].

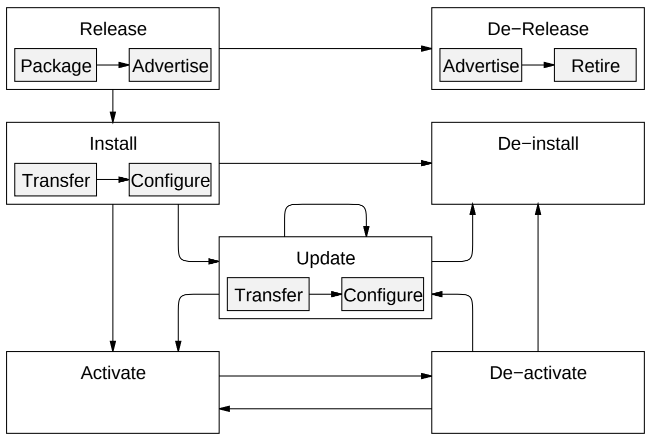

Figure 1: Activities of the Software Deployment Process.

computers administered identically. A *software system* is a coherent collection of artifacts, such as executable files, source code, data files, and documentation, that are needed at a site to offer some functionality to end users. We assume that software systems evolve over time, and that a *version* of a system refers both to time-ordered revisions as well as to platform specific and/or functional variants.

A *resource* is anything needed to enable the use of a software system at a site. Examples include IP port numbers, memory, disk space, as well as other systems. Some resources (e.g., data files) may be sharable, while others may be used by only one system at a time (e.g, IP port numbers).

Deploying a software system involves the transfer or copy of its constituents from a *producer site* to one or more *consumer sites*, which are the target of the deployment process. Once deployed, a software system is *available for use* at the consumer site.

2.3 The Software Deployment Process

It should be evident from the motivating scenario presented above that software deployment is a complex process in its own right. The deployment process consists of several interrelated activities, as depicted in Figure 1. Arrows in the figure indicate possible transitions between activities for some particular deployed system, suggesting a sort of lifecycle model for a system in the field. No assumption is made, however, about the location where activities are carried out; they can occur at the producer site (e.g., releasing a patch) or at the consumer site (e.g., removing a component), or at both (e.g., purchasing a component).

Although we can identify a set of distinct and key activities that typically constitute a generic deployment process, we cannot precisely define the particular practices and procedures embodied in each activity. They heavily depend on the nature of the software being released, and on the characteristics and requirements of the producers and consumers. Therefore, Figure 1 should be interpreted as a reasonable process that has to be customized and enriched according to specific requirements of the deployment activity being observed. In the remainder of this section we briefly discuss the general characteristics of the software deployment process.

Release. The release activity is the interface between the development process and the deployment process. It encompasses all the operations needed to prepare a system so that it can be correctly assembled and shipped to the consumer site. Thus, the release activity must determine all the resources required by a software system to correctly operate at the consumer site. It must also collect all the information that are necessary for carrying out subsequent activities of the deployment process. This information may be derived from a variety of sources including the development process and the human knowledge about the system structure and operation.

The release activity includes the *packaging* of the system in some form so that it can be transported to the consumer site. This package must contain the system components, a description of the system including the requirements and the dependencies on other external components, the deployment procedures, and all the information that are relevant for the management of the system at the consumer site.

Another step in the release activity is the *advertising*, i.e., the set of operations that are needed to disseminate appropriate information to interested parties about the characteristics and usage of the system being deployed.

Installation. The installation activity covers the initial insertion of a system into the consumer site. Usually, it is the most complex of the deployment activities because it deals with the proper assembly of all the resources needed to use a system. With the term installation we refer to two distinct sub-activities. The first one is the *transfer (or delivery)* of the product from the producer to the target consumer site. The second one consists of all the *configuration* operations that are necessary to make the system ready for activation on the consumer site.

Activation. Activation refers to the activity of starting up those components of a system that must execute to enable the execution of the software system being deployed. For a simple tool, activation involves establishing some form of command (or clickable graphical icon) for executing the binary component of the tool. For a complex system, it might be necessary to start servers of daemons before the software system is launched.

Note that the installation process may itself require the invocation of other tools and, possibly, their own installation. For example, if a system has been packaged as a zip file, the installer must be able to activate the unzip tool to extract the pieces of the system to be installed. If zip/unzip is not available, then a recursive installation may be required to obtain and install the unzip tool.

De-Activation. De-activation is the inverse of activation, and refers to the activity of shutting down any executing components of an installed system. In general, de-activation is required before other deployment activities –e.g. update– can commence.

Update. The process for updating a version of a system usually is a special case of installation. It is usually less complex because it can often rely on the fact that many of the needed resources have already been obtained during the installation process. Typically the deployment life-cycle includes a repeated sequence in which a system is de-activated, a new version is installed, and then the system re-activated. For some systems, de-activation may not be necessary and update can be performed while a previous version is still active.

Similarly to installation, update includes the transfer of all the components needed to complete the operation.

De-Installation. At some point, a system as a whole is no longer required at a given consumer site, and may be removed. We assume that de-installation is preceded by de-activation. The de-installation activity possibly involves some re-configuration of other systems in addition to the removal of the files belonging to the system that must be de-installed. De-installation is not necessarily a trivial process. One reason is that the de-install process may assume that it is the only user of some shared resources such as various data files, and so it may release those resources and cause other software systems to fail.

Obsolescence. Ultimately, a system is marked as obsolete and support by the producer is withdrawn. As with de-installation, care must be taken to ensure that the withdrawal will not cause difficulties. This requires that the withdrawal be advertised to all known consumers of the system.

2.4 Software Deployment Issues

The scenario presented in Section 2.1 has illustrated typical problems encountered in deploying software systems. The terminology defined in Sections 2.2 and 2.3 makes it possible to refer in a consistent and comprehensive way to the different activities and procedures that constitute the software deployment process. In this section, we will take advantage of the concepts and examples introduced in the previous two sections to raise a series of issues and problems that characterized software deployment. They have been important factors that have guided our definition of the evaluation framework.

2.4.1 Change Management for Installed and Running Systems

The evolution of the computerized system is a natural process and it is almost inevitable. It has a significant impact on software deployment activities.

- *Hardware components* can be added or removed. For instance, network interfaces, video cards, various storage devices, and additional memory are added to the system to achieve better performances or support new functionalities.
- *Software components* can be added, updated, or removed. As a consequence of the installation of new hardware, it may be necessary to install new drivers or other new operating system modules. Installing drivers or updating existing ones can be also necessary to access new services or features not provided by older ones or because the new version corrects bugs of the old version. For the same reasons, other software components such as libraries, graphical user interfaces, compilers, and command interpreters can require updates.
- *Software systems* can be added, updated, or removed. New features, better performances, and bug fixing are typical driving factors that motivate some form of change in the set of applications used by an end-user.
- *Running components* can be activated, suspended, reconfigured, or deactivated. These activities are performed to adapt the run-time configuration of a system to new installed components. Also, these activities are part of many ordinary system administration procedures (e.g., back-up, system boot-strap or shutdown etc.).
- *Logical components or common environments* can be added, reconfigured, or removed. By logical components we refer to all those configuration entities that can be shared by applications, e.g., user profiles, shared directory structures, and common disk workspaces.

2.4.2 Dependencies among Components

Managing changes is not difficult per se. Rather it becomes a hard task whenever there are *dependencies* between the numerous components of a managed system.

The term *dependency* denotes any kind of “use” relationship that holds between software components. It is important to distinguish among *classes of dependencies* that are significant for the deployment and management process. In the example of section 2.1, we have stand-alone applications as well as distributed client-server applications. In the case of a client-server application, the client configuration depends on the server configuration and location, and vice versa. Stand-alone applications show a number of dependencies as well that are introduced by the existence of shared components. For example, a presentation tool may require a graphical editor that may be already installed as part of a word processor. Another class of dependencies are introduced when components rely on others components for their installation, activation, initialization of its persistent data, update to a new version, or recovering in case of failures or exceptions. For instance, a system that is distributed with sources files would probably need the `make` utility and a C compiler for its installation. An application that uses the X11 windows system would probably need some shared graphic libraries. A web-based application would probably require an HTTP server running on a well known machine and accessing some specific directories (e.g., for the cgi-bin scripts belonging to the application). An update to a database application may require a special tool that converts old data files in the new format.

In general, as a consequence of good modular design and reuse principles and practice, any software system of a significant complexity is made of separate components with various inter-dependencies. These dependencies among software components significantly increase the complexity of every step of the deployment process.

2.4.3 Large-scale Content Distribution

In the example presented in section 2.1, a major problem is the transfer of data files from the organization headquarter to each branch in a way that minimizes transmission costs. In general, one important step of the deployment process is the transfer of information between different locations of a network. We refer to this activity as *content distribution* or *content delivery.*

This aspect of software deployment becomes significant and requires special attention when the size of the software system and the complexity/dimension of the network scale-up. For example, when large data files have to be sent over slow or unreliable links. Or also when some content has to be deployed over a large number of machines.

2.4.4 Managing Heterogeneous Platforms

The connectivity offered by computer networks and the use of standard communication protocols have made it possible for machines with different hardware and operating systems to cooperate to support distributed applications. In the motivating example, we have mainframes, UNIX workstations and servers, and personal computers running MS-DOS or Windows.

The coexistence and the interoperability of heterogeneous platforms pose new challenges for software deployment. The software that supports deployment has to be ported to every platform. Moreover, the platform type becomes a new variable that has to be taken into account when dealing with configurations and dependencies. For example, a user profile configuration can specify different search paths for executables according to the type of the system the user logs on. Or else, the configuration of a workstation can be adjusted so that it will access some shared library that are specific for its architecture.

In general, dependencies vary across different platforms. For example, consider an application that uses the X11 libraries. Two different platform-specific versions of that application that use dynamically linked libraries will depend on the corresponding platform-specific version of the libraries while in other cases, e.g., on a platform that does not provide dynamic linking, that same application may not require any shared library at all. As a consequence of these differences, almost all the deployment activities must be adapted to take care of platform-specific information.

2.4.5 Deployment Process Customization and Coordination

A significant concern emerging from the example of section 2.1 regards *coordination* and *customization* of the deployment process.

In the example, the business imposes some “non-functional” constraints over the deployment activities, in particular, the timing and schedule of the deployment activity is limited so that those activities don’t interfere with the normal business tasks. In other cases, we can envision a number of system administration policies that are specific for each site that may apply to every deployment activity. For example, for security reasons, the installation policy may require the system to go through a special inspection before it is installed. Alternatively, it may require a trial installation in a safe environment and some test before the system is installed in the production environment. Similar procedures can be required for other activities, for instance, a backup of the old system may be required before a new update or patch is installed.

In general, most deployment activities take place at the consumer site. They make use of system resources and often require exclusive access to system components. Also, they might introduce conflicts with installed or running software. Therefore, the consumer must be enabled to coordinate and customize deployment activities according to his/her own policies and requirements.

2.4.6 Integration with the Internet

The Internet plays a central role in software deployment, in particular with the development of electronic commerce. Dealing with secure distribution, licensing, and billing of software and services is a growing concern because the Internet has created a virtual marketplace worldwide. The connectivity offered by the Internet is providing software producers and consumers with new possibilities of interaction. Producers can advertise their products by making technical specifications and requirements (e.g., dependencies) available on-line. Customers are enabled to evaluate the impact of the installation of a new product and can exploit

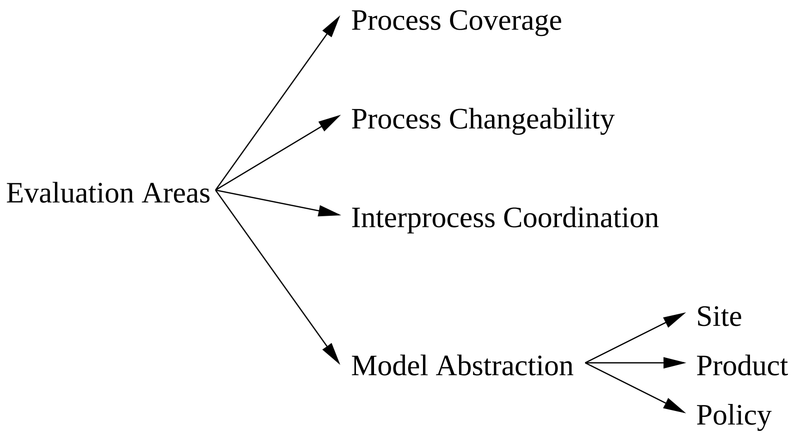

Figure 2: Areas of Evaluation.

the Internet to provide producers with feedback, comments, suggestions, bug reports, usage patterns, and new requirements. In response, producers can deliver on-line support and automatic deployment of patches and upgrades.

2.4.7 Security: Privacy, Authentication, and Integrity

Internet is an insecure network. This fairly obvious observation introduces a variety of issues and concerns. Specifically, there are three aspects of computer security that are particularly critical with respect to software deployment. They are *privacy*, *authentication*, and *integrity* of data.

In the scenario of section 2.1, the headquarter transmits critical information (products information including prices) belonging to the organization over an insecure wide-area network (possibly the Internet). Clearly, in this case, the organization is concerned with privacy, i.e., they want to ship the files to every branch in such a way that nobody outside the organization can read them. As a second example, we have the headquarter accessing some resources at each branch for management purposes. Here, reliable authentication procedures must be in place so that only the authorized system managers can gain privileged access to the server machines at every branch. Finally, the organization interacts with many software providers and establishes automatic maintainance and update procedures. Even if the transfer of software is carried out in a secure way, there might still be security concerns related to the installation of software in the "production" environment. In particular, it is important to guarantee integrity of the organization's data against the execution of malicious or incorrect procedures that may cause corruption or loss of data during installation or update, or during the normal operation of the new version of the system.

3 Evaluation Framework

Having characterized the basic issues associated with software deployment, we now present an evaluation framework for deployment technologies. The framework divides the evaluation into four primary, and largely orthogonal categories, as depicted in Figure 2. Each is explained below. A detailed discussion of the characteristics of deployment technologies that lead to different ratings in each category is left to the survey of existing systems in Section 4.

3.1 Process Coverage

Figure 1 illustrates the richness and complexity of the software deployment process. Each box in the figure represents a significant activity requiring specialized support from a deployment system. This leads to our first evaluation criterion, *process coverage*, which is the degree to which a given deployment system covers each of the constituent activities of the process.

We recognize three values in the evaluation of process coverage: no support for the given activity, minimal support, and full support. When a system does not explicitly recognize or implement one activity as part of its process, we say that there is no support. A minimal support for an activity is provided by those systems that have at least a *hook*, i.e., an access point, to that activity, but do not implement that part of the process. The rating changes to full support for those systems that implement at least a default activity and precisely define its interface so that the activity can be fully integrated in the overall process.

3.2 Process Changeability

The definition of a particular deployment process for a given product and a given target site can be a difficult task. Sometimes the developer of the process, typically someone at the producer site, cannot fully anticipate all the requirements for the process. For example, a consumer may want to insert steps into the process that are intended to perform specialized tests of the status of the deployment. This suggests our second evaluation criterion, *process changeability*, which is an indication of whether the deployment process can be changed after definition.

Here we are concentrating, not on the normal evolution of the process to correct mistakes or respond to modifications to the product, but instead on unanticipated and ad hoc changes typically performed by consumers. An ability to change the process implies an explicit, manipulable representation of the process.

3.3 Interprocess Coordination

Coordination support is required when various deployment procedures of different systems, possibly distributed over a net, have to cooperate and synchronize.

This situation can arise when deploying a composite system that requires and triggers the deployment of the sub-systems it depends on. A typical example is constituted by client-server applications in which the installation or the client has to be coordinated with the installation and activation of the server, and, conversely, where the update of the server has to be coordinated with the de-activation and possibly the update of the client.

As far as coordination is concerned, we will examine the ability to deal with distributed and composite systems. In particular, we will examine whether or not, a system supports synchronization and data exchange between different activities and whether this support is extended over the net.

3.4 Site, Product, and Policy Abstraction

An especially useful way to evaluate deployment technologies is to consider the relative difficulty of creating a particular deployment process from the perspective of the creator of that process. The deployment process can be seen as a procedure that controls the execution order of, and allocation of resources to, deployment activities. This procedure must be instantiated for a particular product and for a particular target site under a particular set of execution policy constraints.[2] The degree to which information about the product, site, and policy can be abstracted out of the procedure allows the procedure to be used in a wider range of situations. This reduces the effort required to create a deployment process and leads to our fourth evaluation criterion, *abstraction*.

We start by considering the "worst" case for deployment, where one would need $S \times P \times A \times X$ different deployment procedures, where S is the number of target sites, P is the number of products, A is the number of different activities covering a complete deployment life-cycle, and X is the number of all possible sets of policy constraints posed for a deployment process. Informally, this case would be similar to having a deployment system consisting of a separate script or "Makefile" for doing some activity—say installation—for every product into every known site with every kind of execution policy.

In Section 2.3, we already defined a set of elements along the direction of the deployment life-cycle, i.e., we defined a number of "values" for the A dimension. Thus, we can immediately factor out A and concentrate on the cross-product $S \times P \times X$.

[2]Recall that a product can consist of multiple components, each of which may be independently developed. Similarly, a target site can consist of multiple, heterogeneous machines, each of which may be managed by independent organizations.

The main idea here is to pull out from the deployment procedures the information related to the deployment process as much as possible and to frame it into three models, each one corresponding to the dimensions defined above, i.e., *site*, *product*, and *policy*. By doing so, we progressively simplify the complexity of the cross-product by factoring out its dimensions. In the ideal case, we will have three orthogonal models that apply to a set of generic deployment functions, one function for each deployment sub-activity. For example, suppose that we have a model that describes a site, e.g, serl.cs.colorado.edu, a model that describes a product, e.g., LaTeX2e, and a model for a particular deployment policy, e.g., root-authorization. Logically, we will be able to define a generic installation procedure in the form: *install(serl.cs.colorado.edu,LaTeX2e, root-authorization)*.

The site model, the product model, and the policy model define a conceptual architecture that we will use together with the deployment process model to evaluate commercial and research systems for deployment in Section 4. None of the system that we examined is designed following this paradigms even though parts of our conceptual architecture can be found in many of them. A prototype that explicitly refers to this conceptual architecture and implements a site model and a product model is presented in [19]. Below, we identify more in details each single model of our conceptual architecture.

3.4.1 Site Model

A site model is a standardized way of describing or abstracting a site's configuration and its capabilities. A site model for a single computer would contain information including the machine type, the operating system, the available hardware resources, the available software resources, etc. This site abstraction, possibly in the form of a standard schema, may then be made available through a standardized interface. The more complete the abstraction, the more a deployment process can be made independent of the target site. In the limit, it should be possible to factor out completely the target-specific information.

Constructing the common consumer site abstraction represents a one-time cost at each site. But once in place, then the construction cost can be amortized across all products to be installed at that site.

This piece of the architecture enables all targets to be treated in the same manner by creating standard methods to query the site's configuration or providing standard mechanisms for performing required tasks. This gives the producer the ability to ignore target site anomalies and to create one procedure for a specific deployment activity, such as installation, for each variant of the deployable system without regard for the target site.

Mechanisms such as Gnu's Autoconf [14] and the Microsoft Registry [7] show how this factoring can be achieved for consumer information, but in two rather different ways. Autoconf is used to produce a single program , "configure", which dynamically computes a system abstraction. The Registry, in contrast, is a passive repository containing the system abstraction. In either case, the deployment process, or more accurately the installation activity, is significantly simplified since a producer can construct installation scripts that are parameterized by common information available from the system abstraction.

3.4.2 Product Model

In order to model the data that has to be deployed, a product (or system) abstraction must also be consistently described. This deployment system abstraction, though similar in purpose to the consumer site abstraction, does not abstract the same type of information. The deployment system abstraction is used to create a full description of the constraints and dependencies of a system to be deployed, such that all deployable systems can be reasoned about in a consistent manner by a specific deployment procedure.

As with target site information, product information is relatively easy to standardize. It includes such things as contact information, dependency specifications, the set of constituent files, and documentation.

As long as software systems are made of one single executable component and with simple inter-object relationships (such as procedure calls), achieving product factorization appears feasible. But for the more complex system composed of multiple, distributed components, the situation is more problematical. Achieving product factorization imposes a significant burden on the developer of systems to architect his system in such a way that its components and their relationships can be easily specified in some standard form.

As a dual counterpart of the process specification, the product abstraction describes the features of a system and its *state* during the deployment life-cycle (see Figure 3). Similarly to the site model, a product

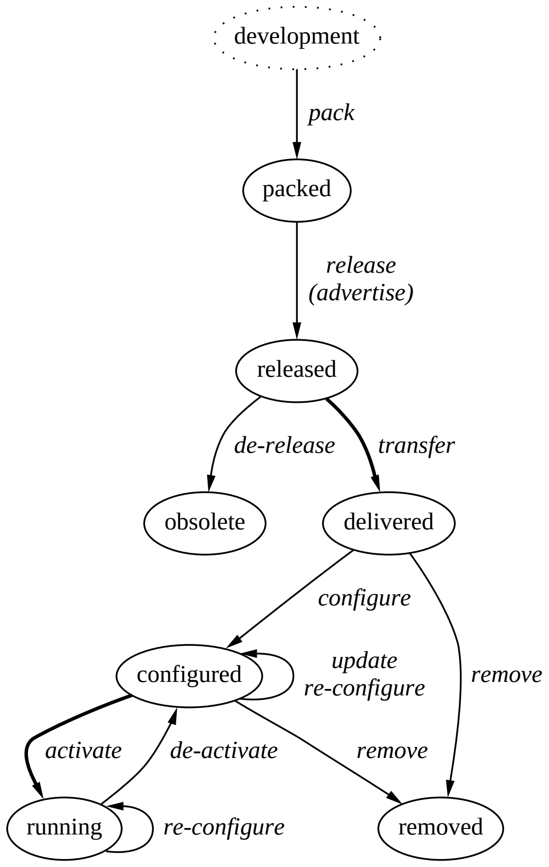

Figure 3: Product/system states: bold arrows represent actions that can be performed multiple times, i.e., “forking” a new system.

model defines a data schema together with some access and query functions so that process activities can use and possibly transform the product meta-data.

3.4.3 Policy Model

We refer to a policy as a particular way of tailoring the execution of a deployment process. For example, consider the installation process for a product that prescribes a sequence of integrity and compatibility checks for the systems on which the product depends that are already installed. One strict policy may require every single check to be performed before the actual installation takes place while a lazy policy may just skip some or all the integrity checks. The term policy is also used to indicate that there is, for example, a generic update procedure, but that it can be parameterized by various criteria such as whether to use a *pull* model versus a *push* model for deciding when to perform an update.

The policy model should then describe all the parameters that can be passed to a deployment activity to adapt its behavior. These parameters include such things as resource usage, security levels, timings, job scheduling, etc.

By analog to the site and product model, we also assume a standard “schema” (structure) for policy models. It is easier to understand this kind of data structure for the other models, where the abstraction really is mostly passive data representing such things as site-specific paths. But here, we are trying to represent procedural information, and the abstraction process is less obvious. In effect, we are hypothesizing the existence of a very high level language in which we can express policies to be applied to generic representations of various activities.

In our rating, as a first approximation in evaluating the policy model, we will consider the parameters that can be passed to the deployment procedures and that affect their behavior. We will also consider the ability to schedule jobs and to allocate or limit resource usage for deployment activities.

3.4.4 Evaluation Criteria

In order to assess the ability of a system to abstract site and product information we evaluate the amount of site and product meta-data that they support and the availability of this information for the process sub-activities. Thus, the factors that determine a good information abstraction are:

- *flexibility* of the data model, i.e., the ability to gapture all the necessary information that can be associated with a site or a product,
- *orthogonality* of the data model, i.e., the ability to separately model different independent aspects and the ability to combine independent system or site attributes,
- *availability* of the data, i.e., the flexibility of the access interface functions or the accuracy of the query functions, and
- *standardization* of the data structure, i.e., the compliance to well known standards and the interoperability with other systems.

4 Current Approaches, Tools and Systems

A wide variety of systems already exist to support various parts of the deployment process, at varying degrees of sophistication, and using a variety of architectures. In the following Sections we discuss many of these tools by giving a summary description, and then characterizing them with respect to our conceptual architecture and our deployment life-cycle. We don't describe each single tool separately, but instead we group them into five classes. Each class is then described presenting the most relevant features among the union of the features of the tools that belong to the class.

We collected the results of this surveys of technologies in two tables. Table 1 presents an evaluation of the systems against the deployment process life-cycle. Table 2 assesses their site and product abstractions and their support for coordination. In the tables, a bullet '•' indicates a full support, a small circle '∘' indicates a scarce support and a void slot indicates no support at all.

4.1 Installers

NET-Install [1], OpenWEB [18] and InstallShield [9] are representatives of a class of systems that are mainly devoted to support software developers or distributors in building packages that can be easily shipped to users and easily installed by an inexperienced end-user on the consumer site.

4.1.1 System Description

These systems are based on a description of the system given by the software producer. The system/package description varies between different systems, but in essence it consists of some global attributes such as product name and version, platform type, default installation directory, required disk/memory space, some contact information etc., plus a simple list of files with a few per-file attributes such as a version number, a digital signature, the encoding/compression algorithm etc.

The package description is used by the packaging systems to build the package file on the producer site. The packaging activity that these systems support is usually very simple: it assumes an almost ready-to-use binary distribution and consists in gathering the files listed in the package description into a single file along with the meta-data specified by the producer.

A fixed installation procedure is also added to the package. The installation procedure, that is executed on the consumer site, uses the meta-data attached to the package and performs a standard sequence of operations: e.g., it checks for the required disk space, it prompts the user for some local information (typically the

		release		install				update			
	System	pack	adv.	trans.	conf.	act.	de-act.	trans.	re-conf.	de-inst.	de-rel.
Installers	NetInstall	○		○	○					●	
	NetDeploy	○		○	○			○	○	●	
	InstallShield	○			●					●	
Packg mgrs	RPM	●		○	●			○	●	●	
	HP-UX SD	●			●				●	●	
Application management	TME-10			●	●	●	●	●	●	●	
	Platinum			●	●	●	●	●	●	●	
	Nebula			●	●	●	●	●	●	●	
	Novadigm				●	●	●		●	●	
Standards	MIF										
	AMS										
	Autoconf				○				○		
Content delivery	Castanet		○	●				●			
	PointCast		●	●							
	Rsync			●				●			
	Rdist			●							

Table 1: Process Coverage

		changeability	coordination	abstraction models		
	System			site	product	policy
Installers	NetInstall				○	
	NetDeploy				○	
	InstallShield	○			○	
Packg mgrs	RPM	○		○	●	○
	HP-UX SD	○		○	●	○
Application management	TME-10	●	○	●	●	
	Platinum	●	○	●		
	Nebula	●	○	●	●	
	Novadigm	●	○	●	●	
Standards	MIF			●	○	
	AMS			●	●	
	Autoconf				○	
Content delivery	Castanet					
	PointCast					
	Rsync					
	Rdist					

Table 2: Changeability, Coordination, and Model Abstraction

default directory) it unpacks or uncompresses the files and possibly it updates or creates some configuration files.

It is usually possible to output either packages for conventional media such as floppy disks or CD-ROM, or for WEB distribution. In the first case, the installation procedure is inserted in the package as a “setup” program. In the second case the package is distributed through the WEB, simply by making it visible on a WEB page. In this case, the installation procedure is bundled with the package in a different way: the system provides an helper/plug-in application that the consumer has to install for his Internet browser that is registered to handle the packages down-loaded by the browser and that implements the installation procedure.

4.1.2 Discussion

Some systems of this class (e.g., OpenWEB) provide a minimal support on the consumer site by keeping a list of installed packages that can be used to support package removal. However, we believe that their process coverage is limited to installation and to a part of the packaging activity.

Note that their main objective is to provide software developers with a packaging utility and a predefined installation procedure to have their products installed. They don’t use or produce product meta-data that can be produced or used by other systems for other deployment activities.

The installation procedure is meant to be more easy, good-looking and user-friendly rather than complete and customizable. Moreover, it can be kept reasonably simple and fixed because these systems mainly concentrate on just one platform (namely Microsoft Windows), this is also why it is possible to ship it in executable format.

These system only support stand-alone products, i.e., products with no dependencies or with a small set of “operating system” dependencies. The site model that they embody is not at all sophisticated because of the assumption about the underlying platform, and is anyway limited to a single machine. They completely lack support for distributed systems and for coordination.

4.2 Package Managers

Almost every modern operating system comes with a suite of utilities that assist system administrators in installing updating and managing software. Examples of such systems are Linux RedHat’s RPM [3], HP-UX SD-UX commands [6] and SUN Solaris *pkg* commands [20].

4.2.1 System Description

These system are based on the concept of *package* and on a local repository that stores the state (the meta-data) of each installed package. A package is an archive that contains files and various information. When a package is installed on a machine, the system records the package meta-data in a local data-base. Any subsequent operation (e.g., query, verification, update or removal) uses the database and keeps its state consistent with the actual installation state.

The main functionalities provided are:

- *create* a package,
- *install* a package, i.e., unpack and configure,
- *query* the installed package database,
- *verify* a set of packages,
- *update* a package,
- *uninstall* a package.

The package meta-data is usually supplied by the producer as a package specification file (PSF or *spec* file). The *spec* file contains a structured list of information in the form (*attribute*,*value*). Attribute-value pairs can be grouped in sub-lists. Here is a brief description of the main groups and attributes[3]:

- *distribution specification*: it refers to the distribution or the collection of tools to which the package belongs. It can contain attributes such as: `title`, `version`, `description`, `vendor` and `copyright`;
- *product specification*: a package may contain one or more products. This group of attributes describes one product. It contains the same attributes (title, version, etc.) found in the distribution group plus various product-specific attributes such as: `machine_type`, `os_name`, `os_version`, `directory` (the default absolute path to the directory in which the product's files will be installed), `category` (e.g., "system-utility", "appl/database", "appl/editor"), `contents`, etc. `contents` is a *file set specification*, i.e., a list of files specifications, each one containing the file name, type, owner and permissions;
- *dependency specification*: it contains any number of products, files or sets of files on which the package depends on. Various kinds of dependencies are supported, e.g., `corequisites` are run-time dependencies while `prerequisites` are install-time dependencies. Dependencies on abstract user-defined capabilities are also supported, i.e., one could specify that the package depends on, say, a mathematical library by specifying `requires=math-lib`. In this case the installation procedure will look for a product or a file-set specification that specifies `provides=math-lib`. It is usually possible to specify dependencies on a set of equivalent packages and, for each package, it is possible to refer to one or more equivalent versions or to a range of versions. E.g., `corequisites=libc,version>=2.0`.
- *control scripts specification*: it contains the names of a number of scripts supplied as part of the package that are called by the package management system during various deployment activities. It contains attributes such as: `checkinstall`, `preinstall`, `postinstall`, `configure`, `unconfigure`, `verify`, `checkremove`, `preremove`, `postremove`, `unpreinstall` and `unpostinstall`.

This information is used by the program that builds the package on the producer side. Once the package is installed, the information from the package and the additional data filled in by the installation/update program is stored in a local database on the consumer site. The procedures of the package management systems have access to this database to use and possibly change the package meta-data. A small set of parameter can modify the behavior of the deployment procedures, typically by forcing or avoiding some internal procedures. In addition, the package management systems provide some query capability that enables any other program, including the package control scripts, to read the system meta-data. The query functions can be applied both to the local database and directly to a package that has not been installed.

4.2.2 Discussion

This class of tools is meant to support system administrators in maintaining the software configuration of a machine or a site. The site abstraction is a simple collection of packages where each package can contain a set of systems and a set of files.

It is important to notice that, despite the simple and generic syntax of the package *spec* file, package manager systems provide a significant added value in abstracting the product information by defining a large set of attributes and assigning them a precise semantics. In facts, the semantics of the attributes is intuitively defined by their names, but more specifically it is determined by the utilities that make use of their values in many phases of the deployment activities. E.g., the `postremove` attribute specifies a script program that is executed by the `uninstall` utility immediately after the files belonging to the package have been removed from the disk.

A large portion of the deployment life-cycle is supported by these systems. They support the packaging activity on the producer site and in some cases (e.g., [3]) this activity is effectively very well integrated with the development environment. They support almost every deployment step on the consumer site except for

[3]This list is basically derived from the HP-UX Package Specification Format [6], but similar concepts and attributes can be found in RedHat RPM [3].

activation, de-activation and content transfer, although some package manager incorporate FTP facilities that allow transparent access to "remote" package files.

Note that Package managers are the only systems that provide some sort of primitive policy model. In facts, they define some parameters that apply to the the basic deployment activities, that allow the system manager to customize them.

There are two major limitations for package management systems. First, they are targeted to a single machine or, at best, for a set of machines that share a network file system, thus, there is very little support for large scale deployment and for distributed systems. Second, because their site abstraction is primarily static, i.e., the site meta-data does not comprise run-time information, these systems lack support for such activities as activation and de-activation and coordination has to be programmed "outside" the system with very little support.

4.3 Application Management Systems

TME-10 [22] from Tivoli, Netview from IBM, OpenView from Hewlett-Packard, IDS from Nebula, EDM from Novadigm are representative of a number of complex, so-called network/applications management systems. Their original purpose was to support the management of corporate LANs. They were capable of detecting hardware failures and network disruptions and reporting them to some operations center for examination. Recently, these systems have ventured beyond hardware and have begun addressing the problems of software management, including some parts of the deployment process.

4.3.1 Systems description

They mostly have a centralized architecture. There is usually a logically centralized "producer", which is some designated central administration site (possibly multi-machine) for all officially approved system releases. In most of them, all the management and deployment activities are controlled by a central management station.

These systems support all the deployment life-cycle at the consumer site. They make use of an explicit definition of the deployment activities that comprises the ability to execute sub-installations on multiple hosts and to coordinate the installation of distributed systems. In addition to the usual deployment activities, they support activation, deactivation and monitoring of applications with the possibility to set up call-back diagnostic routines that respond to anomalies or exceptions generated by the monitoring facility.

All of these systems have a repository, usually centralized, that stores deployment meta-data. The available meta-data can be classified in the following groups:

- *products or packages*: it contains package descriptions (see section 4.2.1), in some cases this repository coincides with the one implemented by a package manager system[4];
- *site configuration*: it stores the static and dynamic configuration of all the machines belonging to the site. Hardware as well as software and run-time information can be stored and retrieved;
- *deployment process specifications*: it stores the deployment procedures. It contains information ranging from installation and update procedures to content deployment plans.

One capability of note is the *inventory*. They are capable of scanning a target site and determining the set of installed systems, and sometimes even the installed version of systems. This information is then brought back to the *configuration* repository.

Another capability of note is their deploy to large numbers of targets. The content deploy can be scheduled and optimized according to the hierarchical topology of the site. Some systems also allow to suspend and subsequently resume part or all the content delivery process.

[4]For example, the HP OpenView system is a sophisticated extension of the HP-UX SD package manager

4.3.2 Discussion

With respect to deployment life-cycle support, these systems support essentially all of the life-cycle activities except the release activities. In some cases they also offer the ability to deal with complex distributed applications (typically client-server applications) in which some coordination is needed to carry out deployment activities. They usually do not support much in the way of producer-side processes and abstractions because they are targeted to relatively "closed" environments in which the producer-consumer negotiation is mediated by the central system management. All the supported deployment activities, including data transfer, can be programmed and coordinated in a distributed environment.

As far as the site abstraction is concerned, the application management systems deliver a detailed model that span the whole site. The model is able to describe a single machine configuration as well as the structure of groups of machines with common properties. The product abstraction is an evolution of the package abstraction that allows to model distributed applications (typically client/server architectures).

4.4 Standards for System Description

As we have seen so far, most of the added value provided by deployment systems consists of well known information structures that model products and sites. As a consequence, the two major key factors for the applicability and interoperability of all the technologies described are the *completeness* of the product and site data models and their *standardization*.

To this end, a number of organizations have proposed various standards for the description of software and hardware components for deployment and management purposes. The Desktop Management Task Force (DMTF) is the major organizational force here and is pushing a standard called the Management Information Format [4, 2] (MIF) for specifying various properties about both hardware and software. Another standardization effort has been made by IEEE with its POSIX standard for software administration [8]. Tivoli's Application Management Specification [23] (AMS) is derived from the DMI. It specifically targets the description of application software systems. The Simple Network Management Protocol [16] (SNMP) defines a standard for defining schemas of information about network components, primarily hardware components. To some extents, GNU Autoconf [13] falls in this category of standard description models. In general, it provides a consumer site abstraction by providing various techniques to determine the consumer site configuration. These techniques include inspecting the consumer site using heuristics and macros or asking a user.

4.4.1 Description

Most of the system description standards are similar to the package description models presented in section 4.2.1, in facts, package managers are based on these standards or they are compliant to their file format and contents.

In general, those standard mandate a special syntax for a system specification file and they define a set of well known values. In other words, they provide a schema, i.e., a typing system or an object model, for system meta-data together with a set of *enumeration* pre-defined types.

4.4.2 Discussion

In terms of our models, these systems and schemas are used to specify both the site abstraction and the product abstractions. Since their primary objective is to model software and hardware component, they are both detailed and flexible to incorporate new elements and structures. Thus, they provide a valuable support in modeling deployment data.

As for package managers, they subsume an articulated deployment process and they provide access point to each deployment activity in the product model although their process modeling capability is far more limited than the data modeling capability.

4.5 Delivery of Content

Marimba's Castanet [15], PointCast, ZIP delivery, rdist[17] and rsync [24] implement content delivery systems. In this class of systems and technologies, the information being deployed is simply transferred from one or more information centers to a number of receiving nodes.

4.5.1 System Description

Point-Cast and ZIP delivery provide news multi-casting services, they are somehow an evolution of the Internet News system [11]. Unlike Internet news, they don't support a bidirectional communication, but they provide news and advertisements through a programmable active receiver application that can be configured to poll the news server for new information. The receiver application has also a library of local display capabilities that enable a graphical presentation of the information that are received.

A consumer can determine which data he wants to receive by subscribing to a number of "channels", possibly from different producers. The subscription or some configuration on the consumer site determines how often or in response to which events the channel has to be updated.

This same publish/subscribe protocol is adopted by Castanet. Castanet is another content deployment system that has some additional features to deal with applications rather than news. A Castanet channel is in essence a set of files. On a regular basis, depending on the configuration of the channel on the consumer site, the consumer pulls an update for that channel. In addition, Castanet enables the producer to customize the channel with a channel *plug-in*, i.e., an application that manages the communication with the consumer and interact with the tuner at the consumer site.

Recent versions of Netscape and Microsoft web browsers introduce new features for content delivery. These facilities convey the idea of an information "push". They allow the user to configure a set of links that mimic a subscription to an information channels. In facts, subscriptions remain local and the browser simply polls and possibly reloads those pages periodically.

Rsync is a file synchronizer that allows to update in an efficient way a set of files. The rsync operation involves two machines, the source machines, that has the files or the new versions, and the target machine, where the set of files must be deployed. Rsync can be invoked in the same way by either the source or the target.

Rdist introduces channel multi-casting at the network and transport layer [10, 25, 12]. Other systems use the standard point-to-point TCP/IP communication primitives and implement multi-casting at the application layer. To do this, they introduce some intermediate repeater stations that concentrate subscriptions and manage some multi-casting locally. This latter approach is adopted by Castanet.

A differential update protocol for transferring files is adopted by Rsync and Castanet. In Castanet, the update operation causes the server to send over to the consumer only the files of the channels that have indeed changed. Similarly, rsync transfers only the blocks of files that differ on the consumer site, thus, performing an incremental update with finer granularity and more efficiency.

4.5.2 Discussion

This class of systems tackles the problem of large-scale content distribution. Information delivery may be treated as the simplest possible form of installation in which no target or product specific computation is carried out; just the rather mindless insertion of data into some point in the consumer side file system.

In any case, the data sent by the server is opaque for the receiver, i.e., there is no configuration or meta-data in the information flow and, other than for some particular functionalities[5], the data are not interpreted by the receiver application at the consumer site.

Hence these systems can be considered as specific technologies to support the transfer activity in the deployment life-cycle. They don't provide support for other activities in the deployment process nor in abstracting product or site information.

From the point of view of our models, these systems are simple, but it is worth noting that they have adopted quite different technologies for carrying out the data transfer function with an eye to making

[5]Some systems provide automatic update of the receiver, i.e., the producer can push an update using the same channel it uses to deliver the content.

them scalable and efficient, especially in a low-bandwidth or costly network environment. They achieve an efficient exploitation of the communication network using two strategies. The first one consists in optimizing a one-to-one communication by shipping "deltas" while the second one applies to one-to-many deployment sessions and it is based on multi-cast transport protocols provided either by the network-layer or by the application-layer.

5 Conclusions

In this paper we discussed the problem of software deployment, a part of the larger software life-cycle that has been given little attention by the software engineering community to date. The processes underlying software deployment are complex and must be better understood in order to produce better, faster, and more automated approaches to the whole deployment life-cycle.

To further our understanding of deployment, we have defined some terminology, and then defined a three part model that provides a framework for evaluating software deployment technologies. This model consists of a deployment process specification, a conceptual architecture and a set of criteria to evaluate existing technologies against the conceptual architecture.

Using this model, we have surveyed a set of representative technologies that are currently available, either commercially or from researchers.

The result of the evaluation suggests several lines of research. First, the models themselves should be deepened to give better characterizations of the problems surrounding deployment. In particular, a better understanding in the area of policy and coordination modeling has to be achieved. Second, we can find no instance of a deployment system that supports every deployment sub-process and there is no evidence of the possibility to integrate some of those systems easily to cover the whole spectrum of activities in the deployment life-cycle. Finally, through the evaluation of state-of-the-art deployment technologies, we proved that the conceptual architecture that we defined provides a good framework to study software deployment issues and to design systems that support software deployment.

Acknowledgments

This study has been carried out together with Alexander L. Wolf, André van der Hoek, Dennis Heimbigner, and Richard S. Hall at the Department of Computer Science of the University of Colorado at Boulder. The initial discussions and ideas have been strongly influenced by the Software Dock [5], an agent-based architecture for automated software deployment.

References

[1] NET-Install, 1996. http://www.twenty.com.

[2] Desktop Management Task Force Inc. *Desktop Management Interface Specification*, March 1996. http://www.dmtf.org/tech/specs.html.

[3] Marc Ewing and Erik Troan. The rpm packaging system. In *Proceedings of the First Conference on Freely Redistributable Software*, Cambridge, MA, USA, February 1996. Free Software Foundation.

[4] Desktop Management Task Force. Software standard groups definition, November 1995. http://hplbwww.hpl.hp.com/people/arp/dmtf/ver2.htm.

[5] R.S. Hall, D.M. Heimbigner, A. van der Hoek, and A.L. Wolf. An Architecture for Post-Development Configuration Management in a Wide-Area Network. In *Proceedings of the 1997 International Conference on Distributed Computing Systems*, pages 269–278. IEEE Computer Society, May 1997.

[6] Hewlett-Packard Company. *HP-UX Release 10.10 Manual*, November 95.

[7] Jerry Honeycutt. *Using the Windows 95 Registry.* Que Publishing, Indianapolis, IN, 1996.

[8] IEEE Standard for Information Technology. *Portable Operating Interface System–Part 2: Administration (POSIX 1387.2)*, 1995.

[9] InstallShield, 1997. http://www.installshield.com.

[10] J. Ioannidis and G. Maguire Jr. The coherent file distribution protocol. RFC 1235, June 1991.

[11] Brian Kantor and Phil Lapsley. Network news transfer protocol, a proposed standard for the stream-based transmission of news. RFC 977, February 1986.

[12] John C. Lin and Sanjoy Paul. Rmtp: A reliable multicast transport protocol. In *Proceedings of the IEEE INFOCOM '96*, pages 1414–1424, March 1996. ftp://gwen.cs.purdue.edu/pub/lin/rmtp.ps.Z.

[13] D. Mackenzie, R. McGrath, and N. Friedman. *Autoconf: Generating Automatic Configuration Scripts.* Free Software Foundation, Inc, April 1994.

[14] David MacKenzie. *GNU Autoconf.* Free Software Foundation Inc., March 1995.

[15] Marimba Inc. *Castanet White Paper*, 1996. http://www.marimba.com/developer/castanet-whitepaper.html.

[16] K. McCloghrie and M. Rose. Management information base for network management of tcp/ip-based internets. RFC 1156, May 1990.

[17] Daniel Nachbar. When network file systems aren't enough: Automatic software distribution revisited. In *Proceedings of the USENIX 1986 Summer Technical Conference*, pages 159–171, Atlanta, GA, June 1986. USENIX Association.

[18] OpenWEB netDeploy, 1997. http://www.osa.com.

[19] Richard S. Hall and Dennis Heimbigner and André van der Hoek and Alexander L. Wolf. The Software Dock: An Architecture for Post-Development Configuration Management in a Wide-Area Network. Technical Report CU-SERL-202-96, University of Colorado Software Engineering Research Laboratory, Boulder, Colorado 80309-0430, 5 October 1996.

[20] SUN Microsystems. *SunOS 5.5 Manual*, September 92.

[21] Owen H. Tallman. Project Gabriel: Automated deployment in a large commercial network. *Digital Technical Journal*, 7(2):52–70, October 1995.
http://www.europe.digital.com/.i/info/DTJI05/DTJI05SC.TXT.

[22] TME/10 Software Distribution. http://www.tivoli.com/products/Courier.

[23] Tivoli Systems Inc. *Applications Management Specification*, 1995.
http://www.tivoli.com/products/tech_info/AMS/AMS.html.

[24] Andrew Tridgell and Paul Mackerras. The rsync algorithm. Technical Report TR-CS-96-05, dcs-anu, anu-address, June 1996.

[25] Brian Whetten, Todd Montgomery, and Simon Kaplan. A high performance, totally ordered multicast protocol. In A. Schiper K.P. Birman, F. Mattern, editor, *Theory and Practice in Distributed Systems.* Springer Verlag, 1995.